9. Les structures répétitives

Définition

Une structure répétitive (ou *itérative*, appelée aussi *boucle*) oblige le programme à répéter un bloc d'instructions jusqu'à ce qu'une condition booléenne soit fausse.

Etude de cas : Somme des entiers de 1 à 10.

On se propose de calculer la somme des nombres entiers entre 1, 2, 3, …, 10.
Il nous faudra donc une variable qui sera nommée ***compteur*** pour faire le comptage de **1** à **10**, et une autre ***som*** pour faire la somme.
Code sans structure itérative :

```
byte compteur = 1, som = 0;

som += compteur;
compteur++; // compteur vaut 2

som += compteur;
compteur++; // compteur vaut 3

som += compteur;
compteur++; // compteur vaut 4
…
…
…
som += compteur;
compteur++; // compteur vaut 10
Console.WriteLine("valeur de la somme : " + som);
```

10 fois.

La partie répétée sera exécutée tant que **compteur <= 10**
Bien sûr, il n'est pas pratique de répéter ces instructions, d'où l'utilité des ***boucles***.

1- La boucle while

Syntaxe de base :

```
while(condition booléenne)
{
      instructions
}
```

- De même que les autres structures déjà étudiées (if, else, …), si un bloc **while** ne contient qu'une seule instruction, alors les accolades ne sont pas obligatoires.
- Un bloc peut lui aussi contenir des structures alternatives (if, if/else, switch, …), itératives, …
- La boucle **while** continue de s'exécuter tant que la condition booléenne est vraie. Si la condition booléenne est fausse, les instructions de la boucle ne seront pas exécutées.
- Il est essentiel que la condition booléenne devienne fausse après un certain nombre d'itérations, sinon on parle de « boucle infinie ». Une boucle infinie conduit à un plantage du programme.
- L'utilisation de l'instruction **break** permet d'arrêter une boucle, **même si la condition booléenne est encore vraie**.
  Syntaxe de base :

```
while(condition booléenne)
{
      instructions
      if(autre condition booléenne)
```

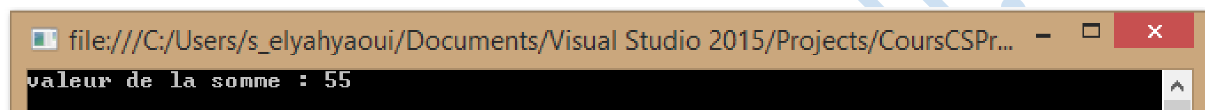
```
            {
                  break;
            }
      }
```

Etude de cas : Somme des entiers de 1 à 10.

```
byte compteur = 1, som = 0;

while (compteur <= 10)
{
    som += compteur;
    compteur++;
}

Console.WriteLine("valeur de la somme : " + som);
Console.ReadKey();
```



2- La boucle do-while

Syntaxe de base :

```
do
{
      instructions
} while(condition booléenne);
```

- La boucle **do-while** a le même effet que la boucle **while**, sauf que la boucle **do-while** vérifie la condition d'itération à la fin et non au début. Donc même si la condition est fausse, les instructions seront exécutées au moins une fois.
- La boucle **do-while** est toujours suivie d'un point-virgule.

Etude de cas : Somme des entiers de 1 à 10.

```
byte compteur = 1, som = 0;

do
{
    som += compteur;
    compteur++;
} while (compteur <= 10);

Console.WriteLine("valeur de la somme : " + som);
Console.ReadKey();
```

2- La boucle for

Syntaxe de base :

```
for(initialisation; condition booléenne; incrémentation/décrémentation)
{
      instructions
}
```

- La boucle **for** a le même effet que la boucle **while**.
- La déclaration de la boucle **for** doit **obligatoirement** contenir ces **3** éléments :
  - **L'initialisation du compteur**,
  - **Condition booléenne de répétition de la boucle,**

- **L'instruction d'incrémentation ou décrémentation du compteur**.

- Le compteur des itérations peut être déclaré lors de la déclaration de la boucle **for**.

Etude de cas : Somme des entiers de 1 à 10.

```
byte som = 0;

for(byte compteur = 1; compteur <= 10; compteur++)
{
    som += compteur;
}

Console.WriteLine("valeur de la somme : " + som);
Console.ReadKey();
```

Exercice 1

Écrire un programme qui lit **2** entiers **X1** puis **X2**, et qui calcule la somme des entiers compris entre **X1** et **X2**. (La valeur de **X1** doit être inférieure à celle de **X2**, sinon on les échange).

Exercice 2

Écrire un programme qui lit un entier **N**, et qui calcule la somme des entiers **pairs** compris entre **1** et **N**.

Exercice 3

Écrire un programme qui lit un entier **N** et qui affiche les entiers de **1** à **N**, **5** par **5**, séparés par des tabulations.
Exemple d'exécution pour **N = 22** :

1 2 3 4 5
6 7 8 9 10
…
21 22

Indication : Utiliser l'opérateur *modulo* (%)

Définition : Boucle imbriquée

Une boucle est dite **imbriquée** quand elle se trouve à l'intérieur d'une autre boucle.
Exemple :

```
while(condition_1)
{
    do
    {

    } while(condition_2);
}
```

Exercice 4

Écrire un programme qui lit un entier **N** et qui produit l'affichage suivant (en escaliers) :

```
1
 2
  3
   4
………
     N
```

Exercice 5

Ecrire un programme qui lit un entier **N** et qui affiche un carré d'étoiles contenant **N** lignes, dont chacune contient **N** étoiles :

```
* * * *  … *
* * * *  … *
* * * *  … *
…
* * * *  … *
```

Exemple d'exécution :

file:///C:/Users/s_elyahyaoui/Documents/Visual Studio 2015/Projects/CoursCSPr...

```
tapez la valeur de N :
5
*****
*****
*****
*****
*****
```

Ensuite on se propose de réaliser les figures suivantes :

```
*
* *
* * *
* * * *
```

Ou bien :

```
* * * *
* * *
* *
*
```

Ou bien :

```
      *
    * *
  * * *
* * * *
…
```